AMOS User's Manual

Responsible person: Manuel Carro, José M. Gómez, Jesús Correas, José F. Morales, Edison F. Mera, Germán Puebla, Daniel Cabeza, Francisco Bueno, Carlo Daffara, and Manuel Hermenegildo, Technical University of Madrid
({mcarro,german,bardo,bueno,herme}@fi.upm.es, {jgomez,jcorreas,jfran,edison}@clip.dia.fi.upm.es, cdaffara@conecta.it)

1 A Gentle Introduction to AMOS

This corresponds to a short, easy-to-read introduction to the tool, why it is different from other search tools and what advantages Amos brings in with respect of other search tools. It also justifies some general design decisions.

The world of Free/Libre Open Source Software (FLOSS for short) has seen a growth that is unbelievable just a few years ago. When the popular FreshMeat.net site was launched, there were just a few updates per day. Now, usually there are more than 50 new software announcements per day, and this represents just a small percentage of the true number - just consider SourceForge and its more than 60.000 projects, of which a large number are at the "beta" or better stage, or all the specialized software packages developed in the scientific and research communities.

As much as it seems a wonderful thing, it is becoming a problem in itself just finding what you are looking for - not only for users, but especially for developers and system integrators that may be willing to cooperate with an existing project, rather than starting something new from scratch. This is especially complex when someone wants to search for specific capabilities that may be embedded in a package, but that are not apparent from the documentation (that may be nonexistent to start with). And if it is a problem now, imagine what will happen in a few years, if the growth in FLOSS software production continues at this pace!

As members of the IST-sponsored AMOS project, we are trying to devise a potential solution to the search problem, through the development of a specialized search engine devoted to searching software code and other code-related artifacts (like code snippets, test cases and such). We are trying to combine several technologies to improve on existing techniques:

- We are using a sophisticated, extremely efficient Prolog environment (called Ciao Prolog, and itself released under the GPL) to be able to perform complex manipulation of symbols,
- The search engine itself is using a structured approach to representing software packages and their relationships, in the form of a very simple "ontology" and a dictionary of potential search terms.
- The algorithm used makes it possible to search for "assemblies", or set of packages that together try to match the user's requirements as closely as possible.

We will try to explain everything in a question-and-answer way. Stay tuned!

So, you say that you are trying to create some sort of search engine. But I can search things on FreshMeat, or SourceForge, or Google! Why another one?

Because searching isn't that simple. Let me make a small example: suppose that you want to find a way for creating a UDF filesystem for pressing a DVD. Let us try to find it with FreshMeat leads us to the results in Figure 1.

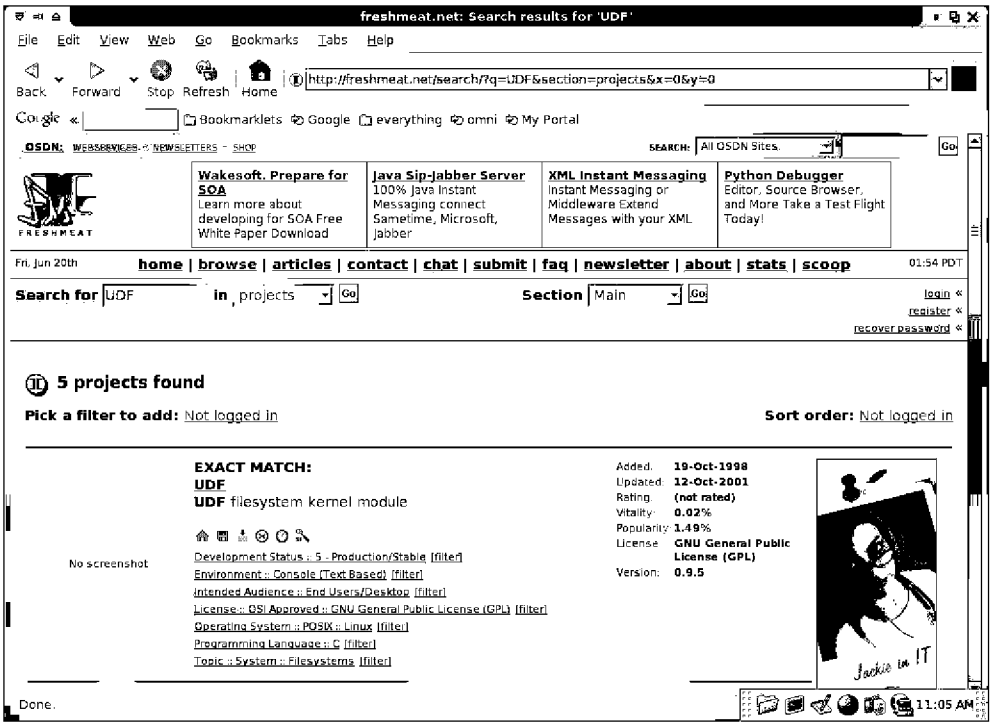

Figure 1: Searching with FreshMeat

But we want to create the image, not read it. The other projects (Figure 2) are not much better, and Google (Figure 3) does not help a lot either.

If you know where to look, you can find the right project (that is on SourceForge, and is called dvd-create). Of course, this is just an example- if we look for a LaTeX-to-PDF converter, for example, FreshMeat returns sensible answers at the 15^{th} and 16^{th} place, while Google fares much better. More complex examples are much more difficult to search for; even more so in specialized areas where word meaning can be different.

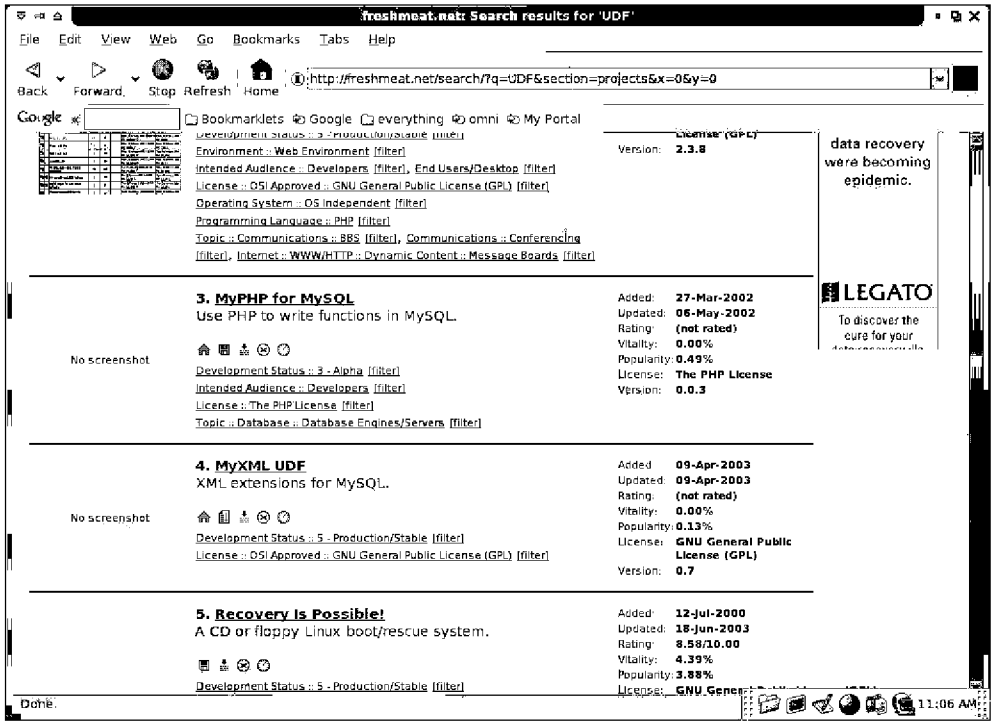

Figure 2: More FreshMeat Projects

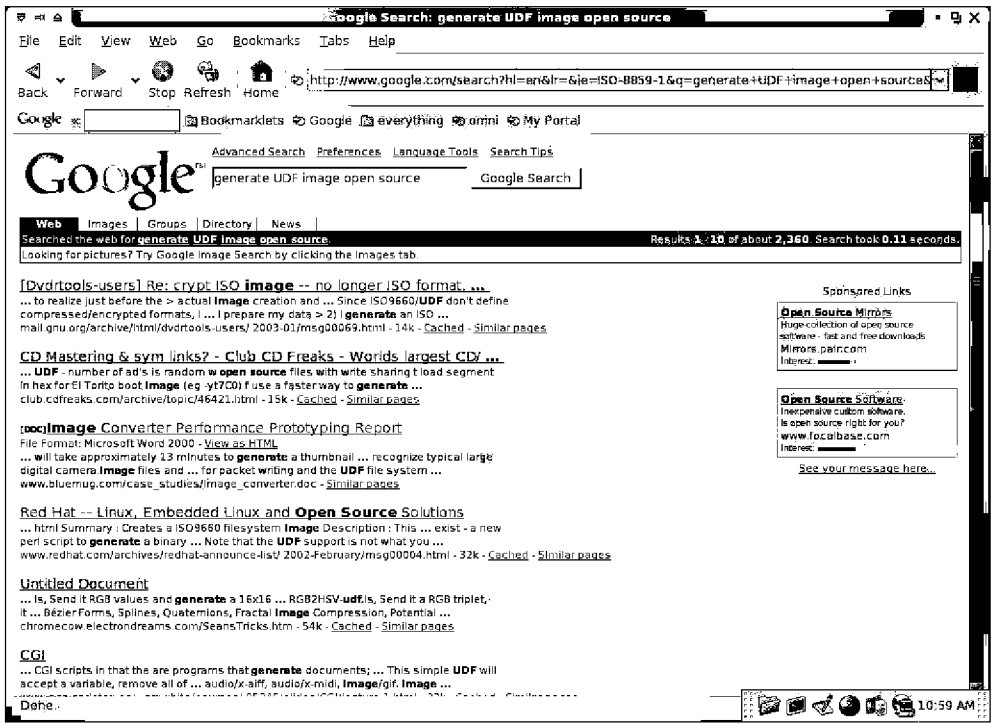

Figure 3: Google results

How do you plan to do it better?

We try to improve the situation by describing packages through their capabilities. That is, we are introducing a set of words (called "dictionary") and using those to describe what the package do. The AMOS search engine is also capable of combining packages together; let me make another example: suppose you want to find a way for converting LaTeX to PDF. You can do it in one sweep, or converting latex to postscript and the postscript to PDF, or using other intermediate formats. AMOS tries to find all suitable chain of packages, and creates "assemblies" that implements (if possible) what is requested.

If it does all this work, it will be too slow to be useful, especially if you imagine to have thousands of packages!

We have tested the engine, by loading all the package descriptions of a RedHat distribution, and using the dependencies as capabilities. This test used 700 package descriptions and around 90000 relationships (dependencies) among them, with a dictionary of some 14000 terms (much more than we expect in a real database to have). This should give a reasonable approximation of the kind of load that may end up having. On a fairly standard PC quite complex queries with all the data stored in a relational database give back results in a few milliseconds (faster than RPM, taking into account that dependencies are used recursively). More detailed information, including detailed figures, can be found in the technical report The Matching Engine Design.

One general reason for this speed is the use of a technique termed inverted indexes, which, so to speak, precomputes part of the search work. This has been classically used in relational databases, and is also in the heart of the Google web search tool.

OK, let's say that we have thousands of packages, and our search returns several possible assemblies. How can we sort through it?

We added to the engine the capability of using heuristics and it is designed so that these can be modified or added easily. For example, it should be possible to prefer sets of packages that are written in the same language, or set of packages with the minimum size. We plan to add to the engine a series of sensible preference rules in order to guide the search and not to overwhelm the user with many undesired matches. As an example, we will add to the engine a special "license" part that will warn when assemblies made of GPL and non-GPL software are mixed together, to prevent potential licensing problem. This is only a warning, since for example mixing user-level packages is

always allowed.

You target FLOSS software. What about you?

The Ciao Prolog system is already Open Source software. The AMOS engine is also distributed under the GPL, and the database and tool information that will be filled up will be under the Free Documentation License (FDL).

But, Prolog?? Why not ¡python, C, C++, Java, name-your-favorite here..¿

Modern Prolog compilers (Ciao Prolog among them) are extremely efficient, and Prolog data structures are perfectly suited for the task at hand. Also, the declarative nature of Prolog maps extremely well to our algorithm, and there are several nice libraries for generation of web pages and interfacing with databases. If you really, really want it, the Ciao compiler can output C as its target, so you can have C if you really want (not that it's much readable, and it will not give you more capabilities than the Prolog system itself.)

OK, I bought it. In one of your technical papers (The Internal Query Language Design) you present also an API (Application Programmer's Interface) to directly call the engine. Is it only for Prolog?

No. As a direct inheritance from Ciao Prolog, the engine can be called from Java and C, and more interfaces are on their way.

What about XML? I like XML!

XML is nice as a representation language; it is quite easy to take the database engine dump (that is, the full content of the database) and output XML. Reasoning over XML is still a difficult problem, especially in terms of efficiency when the XML tree is huge. So, we prefer to do all the reasoning and the internal representation in the Ciao internal format, and leave to output plugins the task of converting to-from other formats.

OK, your project seems interesting. I would like to know something more about how to do searching.

First, a little introduction on the dictionary and the ontology (which, in our case, is just a different way of calling a tree containing all the relations between packages and

the words that describe them). The dictionary is a long list, containing many different "words", or dictionary atoms. For example, "image_blurring" may be a suitable "word" (even if it is composed by two English words). To a dictionary atom we can also attach several synonyms, and one generalization- for example, you can generalize "SQL database" into "database". This is used by the engine when no possible match is found, and generalization are used to try to find an approximate solution. The dictionary list is quite important in AMOS, and most of the effort in adding packages to the search engine is really related to finding the best "words" for describing packages. It is also quite important that the words that are added are not already in, thus leading to what we call "dictionary pollution".

"Dictionary pollution"? What's that?

A fundamental point of our project is the fact that the dictionary stays manageable, that is that it does not grow too much. All searches in AMOS are done only through words that are in the dictionary. You can freely use them in the search row, or use the boxes and select among them (eventually using the CTRL and shift keys to perform multiple selections), but you can't perform a free-text search (it will be no different from Google, in that case).

So, it is important to maintain the dictionary size to a minimum, because people performing the search will be forced to sift through the list, at least in the beginning.

How do you plan to avoid that?

By having an administrative interface, that allows for our reviewers to decide if the words are adequate, and eventually suggest alternatives. This administrative work is part of the project, and after the end of the contract we plan to give this administrative task also to the community. We will maintain to our best the system for at least 2 years after the end of the contract, providing connectivity and machines for that, and we hope to be able to donate at least a person to continue the maintenance work. We hope that the system will be adopted, and will be happy to help anyone that wants to deploy it. Everything contributed will be under the FDL, and periodically we will provide dumps of it for download.

How does the administration work?

When you fill in a package, you simply submit it through the web interface. It is then saved in a temporary database, and reviewed for consistency; the reviewer can propose modification, that are sent back to the submitter through email (including comments). When the reviewer gets back to the submission page, she gets back all the filled fields and the comments, and can change it at will and resubmit. If it is accepted, it gets immediately into the database, and can be immediately searched.

Will it always be like that?

If we see that the need for modifications remains small, in the end we can opt for a no-administration system, and simply let the people enter packages directly into the database. This decision will be left for the end of the project, probably. Of course, the source is there - if you want, you can install your own AMOS!

2 Project Goals: A More Technical Description

Reusing Open Source Code

Open source code (OSC) packages will be used to create a database of (loosely defined) components, or pieces of code that implement a specific functionality. The project does not target the strictly defined component [Mau00] market only, but it tries to find a methodology general enough so as to allow the optimal finding of software which matches some desired characteristics. Software is then considered as a white box piece, which will simplify the reuse of the enormous number of open source projects in existence, and increase significantly the number of elements which can be reused. During the project, a methodology and a tool will be devised which can significantly reduce the work needed to identify, from a large number of components, those which can be used to implement a desired functionality. This will greatly help the reuse of existing code and the take up of open source software, by helping and guiding the search process and by making the usage of lesser-known packages easier. The project aims can be decomposed into the more precise following points:

- Finding an ontology that targets source code software packages. These are understood not only as libraries, but also as well-defined, valuable routines and capabilities inside existing source code. This ontology will be used to describe in a portable and coherent way the properties of packages and, if present, of their subcomponents.

- Creating a dictionary of terms and attributes to describe the software packages and their properties (designed to match the ontology).

- Finding a compact, complete, and portable representation for the ontology.

- Implementing a search and matching engine to allow the automatic generation of a list of software pieces to be assembled, starting from a high level description of the needs of the developer. This search engine will use heuristics to guide the search as directly as possible toward a solution and will take into account metrics to express the preferences of the developer. Developing and assessing these heuristics and metrics will also be performed as part of this project.

- Implementing a suitable and persistent database, capable of holding several hundreds of code descriptions, and interfaced to the matching engine.

- Filling in the database with a large enough number of descriptions to be able to test the matching engine with non-trivial problems.

Selecting Code

The selection of code to assemble from the database will be performed through a specialized matching engine, created using a Logic Programming environment (Ciao Prolog [HBC+00, HBC+99]) developed along the last years within the framework of several ESPRIT projects by one of the project partners (UPM), which is itself distributed under OSC license. The database of software components will be initially extracted (and later extended) from the internal archive developed by one of the partners (Conecta s.r.l.), currently holding descriptions of more than 14.000 open source and publicly available software packages, most of them unlisted in the commonly used software search sites like FreshMeat or SourceForge. Both the matching tool and the database will be open-sourced, in order to guarantee the best possible dissemination of the results. A live demonstration system using a selected database will be created within the project. It will only differ from more realistic cases in its size, and the database schema and query engine should therefore be immediately applicable to problems of any size. The ontology will be flexible enough so as to admit changes (specialization or abstraction of some of its parts) to tailor it to specific, local needs (e.g., to match tools and components created inside a single company, or a group of affiliated companies).

3 First Contact with AMOS

The starting page of Amos (Figure 4) gives a short overview of the project and some indications about how the tool can be immediately used. Figure 5 shows a detail of the menus at the left of the page; all the menus in the tool look alike. The topmost box of this menu set changes according to the page in which it appears; the middle one and the bottommost one are permanent.

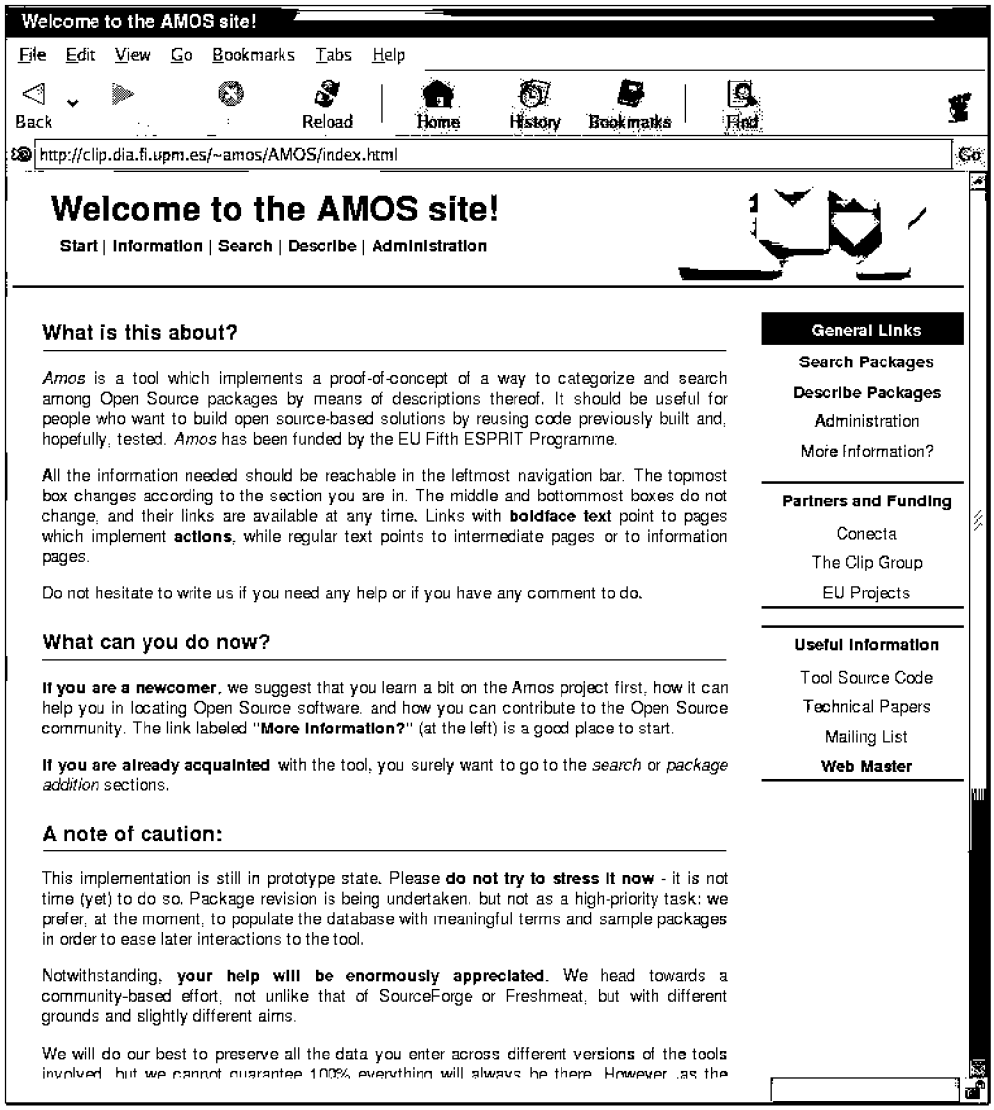

Figure 4: Initial Amos page

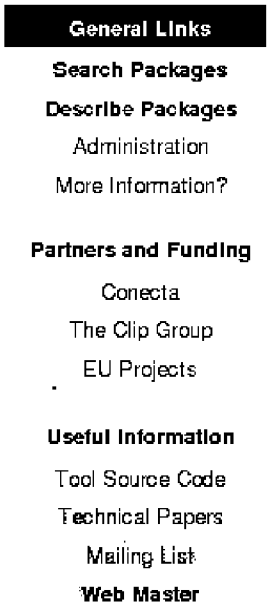

Figure 5: Sample of menus (from the initial Amos page)

What is this about?

Amos is a tool which implements a proof-of-concept of a way to categorize and search among Open Source packages by means of descriptions thereof. It should be useful for people who want to build open source-based solutions by reusing code previously built and, hopefully, tested. Amos has been funded by the EU Fifth ESPRIT Programme.

All the information needed should be reachable in the rightmost navigation bar. The topmost box changes according to the section you are in. The middle and bottom-most boxes do not change, and their links are available at any time. Links with boldface text point to pages which implement actions, while regular text points to intermediate pages or to information pages.

Navigation in the Amos web site is shown in figure 6 as a tree–like structure. The dark shaded state corresponds to the initial point from which all the others can be reached. Leaf states provide no further transition apart from those to the states marked with *, which are reachable from anywhere in the web site at any point of user interaction. These parts of the web site provide access to the fundamental services [GCM04, CGC+04]implemented in the Amos project, i.e. search of packages, insertion of packages into the database, and adminitration of the database and the web site.

The following is a brief description of each state which can be reached from the

initial one, *Start*:

- *General links* provides access to the system main funtionalities:
  - *Search Packages:* Access to the search interface.
  - *Describe Packages:* Access to the packages insertion interface.
  - *Administration:* Access to the system management and package validation facilities:
    - *Review Packages:* Provides access to the package revision process.
    - *Edit dictionary:* Allows inserting, modifying and deleting the dictionary terms contained in the database.
    - *Manage the DB:* Gives access to management services on the database containing validated packages:
      - *Search:* Allows fine search on the package database.
      - *Show Package Information:* Displays all the information provided by the user about the package.
      - *Edit Package:* Allows modifying the package moving the package back to a state pending of validation.
      - *Send to Author for Edition:* Sends an email to the author encoding theURL which gives access to the edition interface.
      - *Delete Package:* Removes the package from the database.
  - *More Information:* Further information about the Amos project.
- *Partners and Funding:* Links to the project members and the EU programme.
- *Useful Information*
  - *Tool Source Code:* Gives access to the *CVS* keeping the project source code.
  - *Technical Papers:* A link to the papers produced during the project.
  - *Mailing List:* A link to the Amos mailing list.
  - *Web Master:* Mail the web master.

Do not hesitate to write us if you need any help or if you have any comment to do.

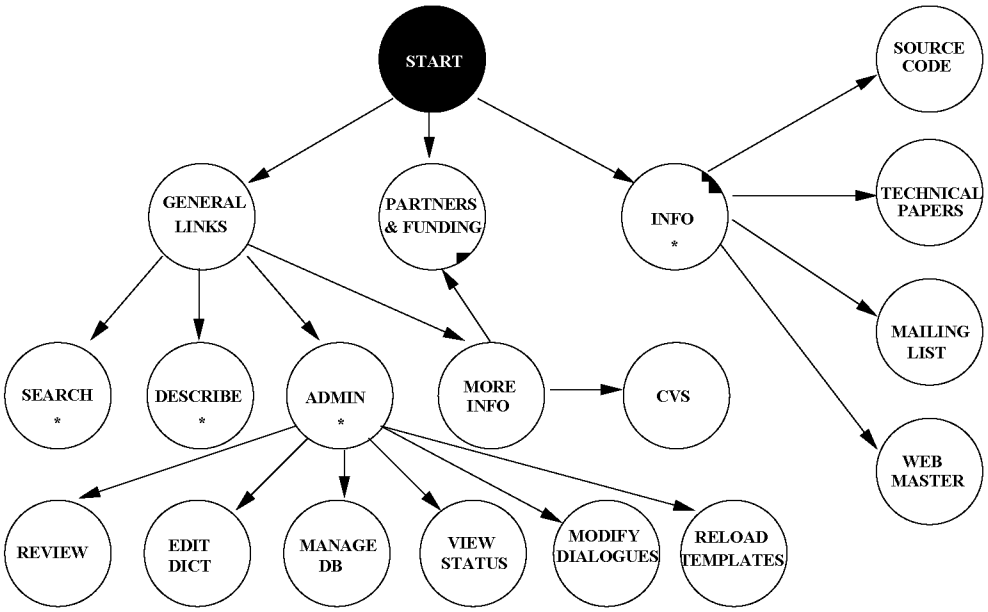

Figure 6: Amos web site navigation

What can you do now?

If you are a newcomer, we suggest that you learn a bit on the Amos project first, how it can help you in locating Open Source software, and how you can contribute to the Open Source community. The link labeled "More Information?" (at the left) is a good place to start.

If you are already acquainted with the tool, you surely want to go to the search or package addition sections.

A note of caution: This implementation is still in prototype state. Please do not try to stress it now - it is not time (yet) to do so. Package revision is being undertaken, but not as a high-priority task: we prefer, at the moment, to populate the database with meaningful terms and sample packages in order to ease later interactions to the tool.

Notwithstanding, your help will be enormously appreciated. We head toward a community-based effort, not unlike that of SourceForge or FreshMeat, but with different grounds and slightly different aims.

We will do our best to preserve all the data you enter across different versions of the tools involved, but we cannot guarantee 100% everything will always be there. However, as the data we enter would suffer the same fate as yours, be sure we will really try to keep it safe.

4 Searching (and Finding)

The search interface (Figure 7) makes it possible to specify characteristics of packages to search for, along with a series of packages the user wants to include (or exclude) from the final solution, and a series of parameters which help (if needed) to cut down the search.

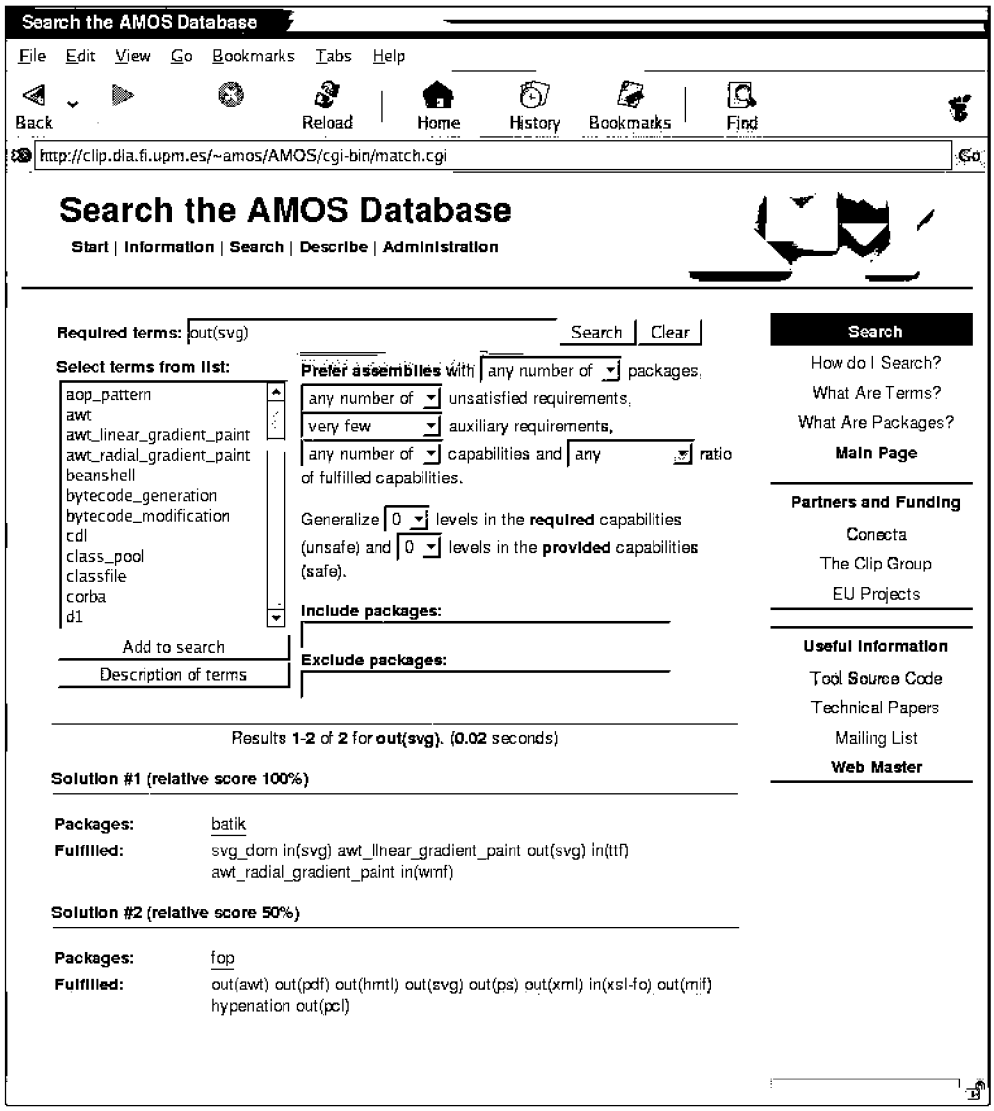

Figure 7: Search interface

How do I Search?

Through the search interface you can find a number of packages fulfilling your software needs. This interface offers a multiple choice menu where the user can select all those terms related with the target software. The terms chosen can be included into the current search either by clicking on the button "Add selected entries to search terms" or just by writing them in the "Search terms" field. It is also possible to set an upper bound to the number of results desired by writing it in the field "Maximum number of assemblies". Additionally, if there is one or more packages required to be part of the result, the user can ensure it by writing the name(s) of the package(s) in the field "Include packages". If, on the contrary, there is one ore more undesired packages the user can write the name(s) in the field "Exclude packages".

Another interesting feature of the search interface is the possibility to select the type of search to perform. It is possible to select the search node and also to specify a number of best branches on which to concentrate search.

Once the desired search parameters are set you only have to click on "Start search" to find a combination of software packages satisfying your needs.

At any time you can also reset the form by clicking on "Revert to initial state" or go through a description of the search terms by clicking on "Description of terms"

What Are Terms?

Search terms are a number of descriptive items provided in order to help the user describe the characteristics of the software sought. Search only attends to these terms and no other will be taken into account. So, it is recommended to choose them among the menu offered by the interface.

What Are Packages?

Open Source Packages are implementations of open source software ranging from libraries to utilities or any other component or piece of knowledge with a clear specification. They can be part of the search itself, but also results are composed by sets of them.

5 Submitting Packages

The package submission procedure takes place currently in a single screen (Figure 8), where the user should describe the package to be submitted. This screen summarizes all the information that is logically spread across a number of entities which are generated from the ontology description [Daf02]. Mandatory fields are placed at the beginning, and links to a description of every field are placed by the editable fields.

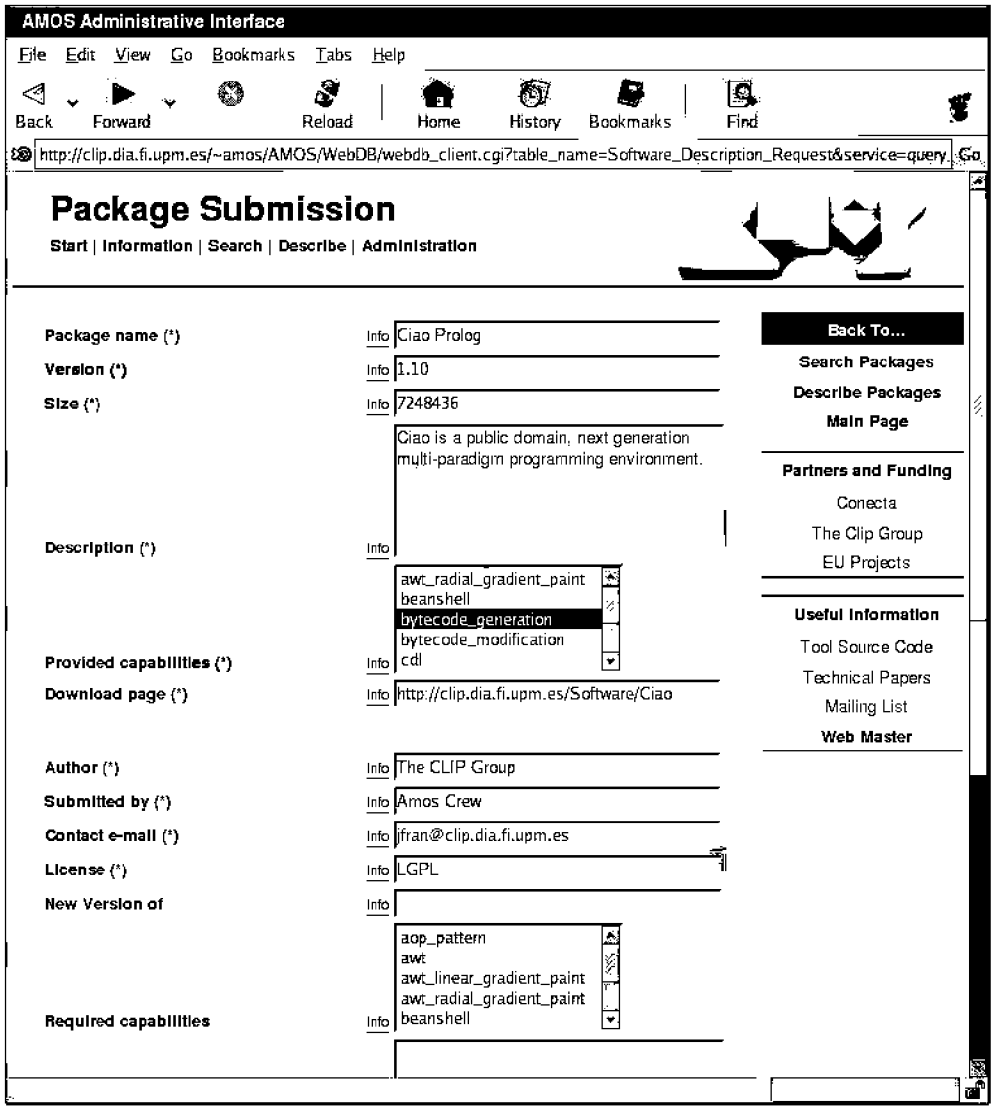

Figure 8: Package submission

Current Submission Fields

Mandatory fields are shown in boldface, and marked with an asterisk in the form.

- **Package Name**: The (unique) name of this package.
- **Version**: What is the version number of this package.
- **Size**: Size of the package (can be used to sort / select packages to be presented to the user).
- **Description**: A terse, text-only description of what the package does.
- **Download page**: The URL from where the software can be downloaded is to be written here.
- **Author**: The name (or nickname) of the original author be introduced here.
- **Submitted by**: Identity of the submitter (i.e., you).
- **Contact e-mail**: This is the email of the person entering the information. It will only be used to get in touch with you regarding the information you have entered.
- **License**: The type of the license (GPL, LGPL, Artistic, Free Documentation, Modified BSD...) applied to the package. Note that in general your package will only be broadly used if you choose a Open Source / Free Software license.
- *New Version of*: If this package is a new version of a previous package, filling in this form will help to track down the version tree.
- *Required capabilities*: Which capabilities are needed by the package.
- *Uses*: Which other facilities are recommended (but not strictly required) to compile the package.
- *Language*: Language the package is written in. May be used to select among different answers to a query.
- *Home page*: If there is any home page where information about the package is stored (such as documentation, mailing lists, etc.) it should be entered here.
- *Creation date*: When the package was created

- *Cost*: Cost of using the package (some packages have an initial cost).
- *Certification*: Type of certification the package has gone through, if any.
- *Security classification*: Type of security classification
- *Maintained by*: This is the name of the maintainer, which can be different from the original author. It helps getting in touch with someone deeply involved in the development of the package.
- *License URL*: A URL where information (e.g., the text) about the chosen license is stored.
- *References*: References of the license.
- *Additional constraints*: Additional license restrictions.
- *Additional freedom*: Additional freedom given by the license.
- *Target environment*: For source packages which have a strong dependency to some operating system or architecture, this field gives information about that dependency.
- *Digital signature*: Digital signature used to verify the package.

6 Accessing and Managing the Database

Administrators may need to access the database, in order to validate packages which have been submitted or to correct or delete packages which are already part of the database, or perform any other maintenance operations. The administrative interface presents a web view of the database and makes it possible to verify manually that the packages meet a minimum of requirements. The interface permits a sophisticated navigation, allowing to search (using regular expressions) and sequentially browse the matching tuples. When reviewing recently submitted packages, the administrator interface is designed to ease the reviewer task by providing tick boxes and a "Reject" button. Rejecting a package will automatically send a message back to the person who entered the package, listing the faulty fields, and giving him/her the possibility to correct them. Additionaly, the interface allows to manage already validated packages, in terms of giving access to package visualization, edition, deletion, and message sending to the owner for edition. Also, the interface facilitates administration tasks on the web site, like reloading the templates which determine the general appearance of the interface, and editing the dialogues.

Manage the DB

The AMOS administrative interface offers a dynamic, context sensitive front-end which allows browsing the packages hosted in the AMOS system. It offers a first page where it is possible to make a package search. All the tuples satisfying the search are displayed in a table, figure 9 with information about the package, namely the package name, its description, and its author. Each entry also presents a number of links to the following services:

Show Package Displays all the information entered about the package entered upon description.

Edit Package Allows modifying the package, and then moving it back to a state pending of validation

Send to Author for Edition Sends an email to the author encoding the URL which gives access to the edition interface.

Delete Package Removes the package from the database.

Reviewing Procedure

The AMOS interface for reviewing packages displays first a list of the packages currently pending of validation, figure 10. Here, a reviewer can click on any of them and be forwarded to a new screen showing the complete description of the package 11. The reviewer can now check the package and decide whether or not to accept it into AMOS, using the above described action "Answer about submission". After doing so, a reviewer can use the current interface to check further pending packages or insert, modify or delete package proposals himself.

Editing the Dictionary

This interface (Figure 12) allows browsing the dictionary terms currently available in AMOS as well as editing, deleting, modifying and enter new ones. The following is a detailed descriptions of these operations:

Sequential search All the results of the query are displayed one by one on demand. The interface provides nice features to browse these results by selecting search state sensitive, dynamically generated search options "Next" and "Previous".

Show matches in a table All the tuples satisfying the query are displayed in a table.

Insert Allows the insertion of a new dictionary term previously checking all the mandatory fields have been filled in. It also ensures the element to be inserted doesn't exist in the table already. If generalizations are provided that correspond to dictionary terms which do not exist in the database, these are inserted into the database.

Delete Deletes all the dictionary terms that instantiate the query. Watch out: Empty form fields instantiate anything!

Modify Allows editing a given entry.

Clean the form After a search action, this option can be used to get an empty form.

Note that when any query is made over a database table, the information written into the form fields is taken as Prolog terms. This way, any empty field of the form is translated into a free variable and therefore will unify all the corresponding arguments of the given table.

Besides, three different search modes are provided:

Normal search mode This is the most restrictive search mode. The results of the query will be just those tuples which exactly match the fields specified.

Case insensitive search mode In this mode upper and lower case is indistinct.

Wildcards search mode Implements a search where wildcards can be used. Special characters are:

***** Matches any string, including the null string.

? Matches any single character.

[...] Matches any one of the enclosed characters. A pair of characters separated by a minus sign denotes a range; any character lexically between those two characters, inclusive, is matched. If the first character following the [is a ĥen any character not enclosed is matched. No other character is special inside this construct. To include a] in a character set, you must make it the first character. To include a '-', you must use it in a context where it cannot possibly indicate a range: that is, as the first character, or immediately after a range.

| specifies an alternative. Two regular expressions A and B with — in between form an expression that matches anything that either A or B will match.

{**...**} groups alternatives inside larger patterns.

' Quotes a special character (including itself).

View System Status

Provides information and statistics about the system usage.

Modify Dialogues

Provides a list with all the dialogues and the text settings of the adminstration interface and allows the system administrator to edit them.

Reload Templates

In case the configuration files have been changed, this option reloads them and updates the interface with the new configuration.

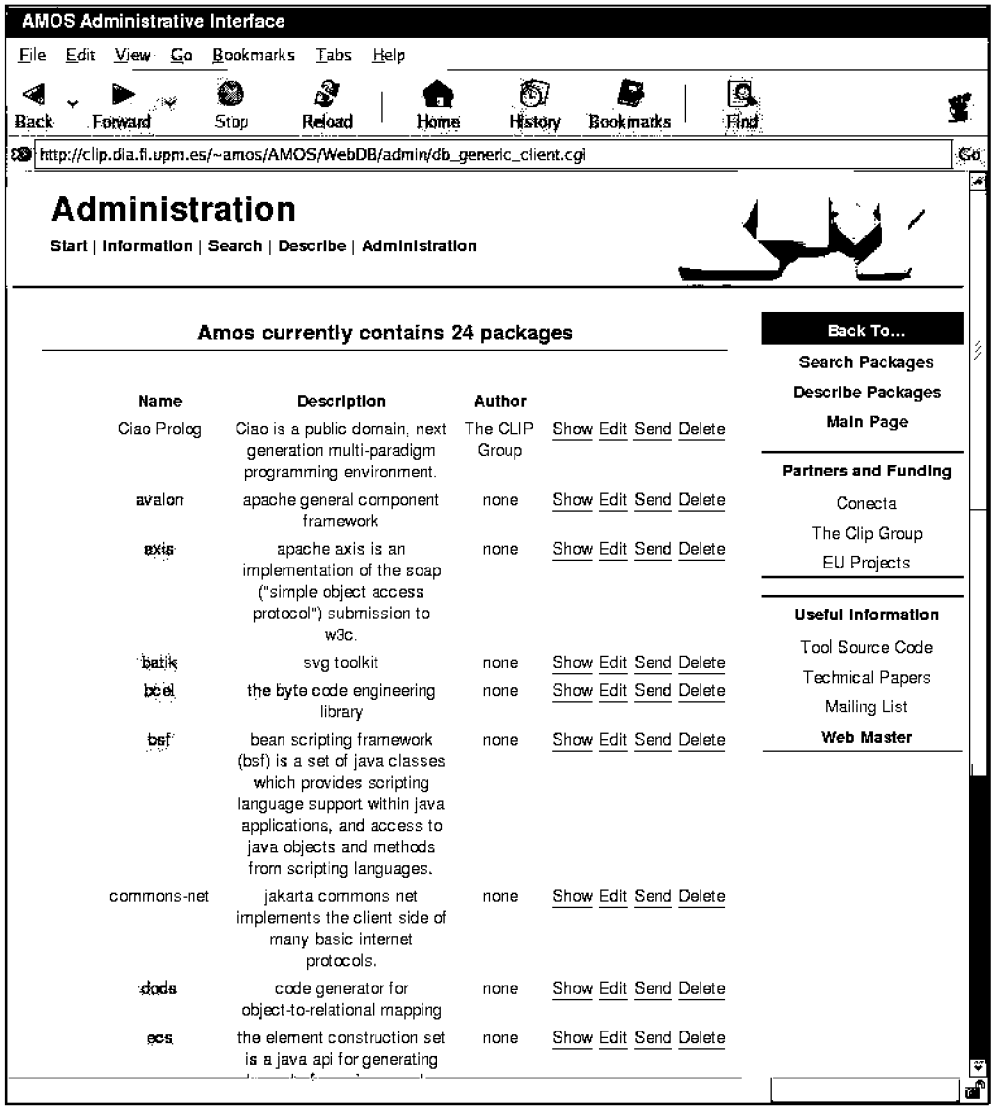

Figure 9: Package Administration

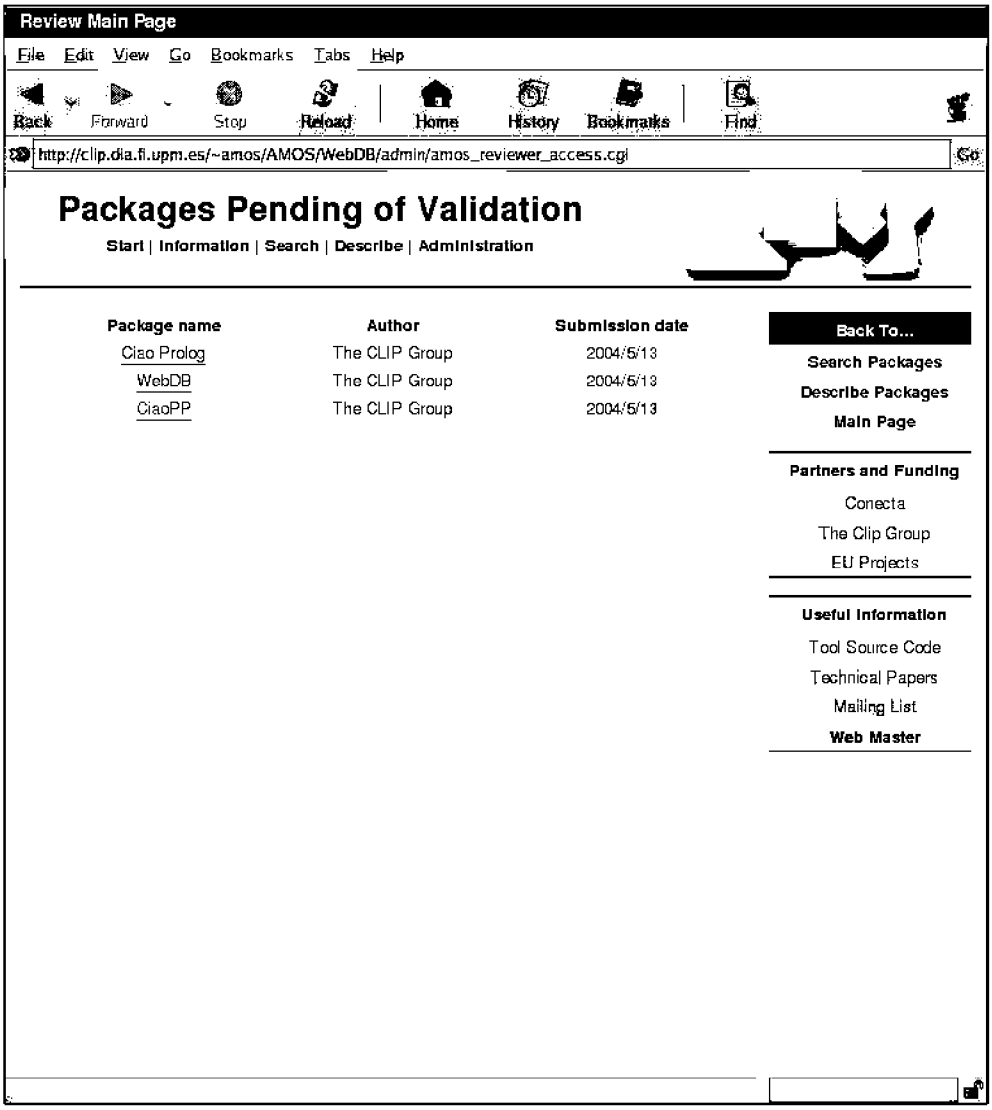

Figure 10: Package validation firt screen

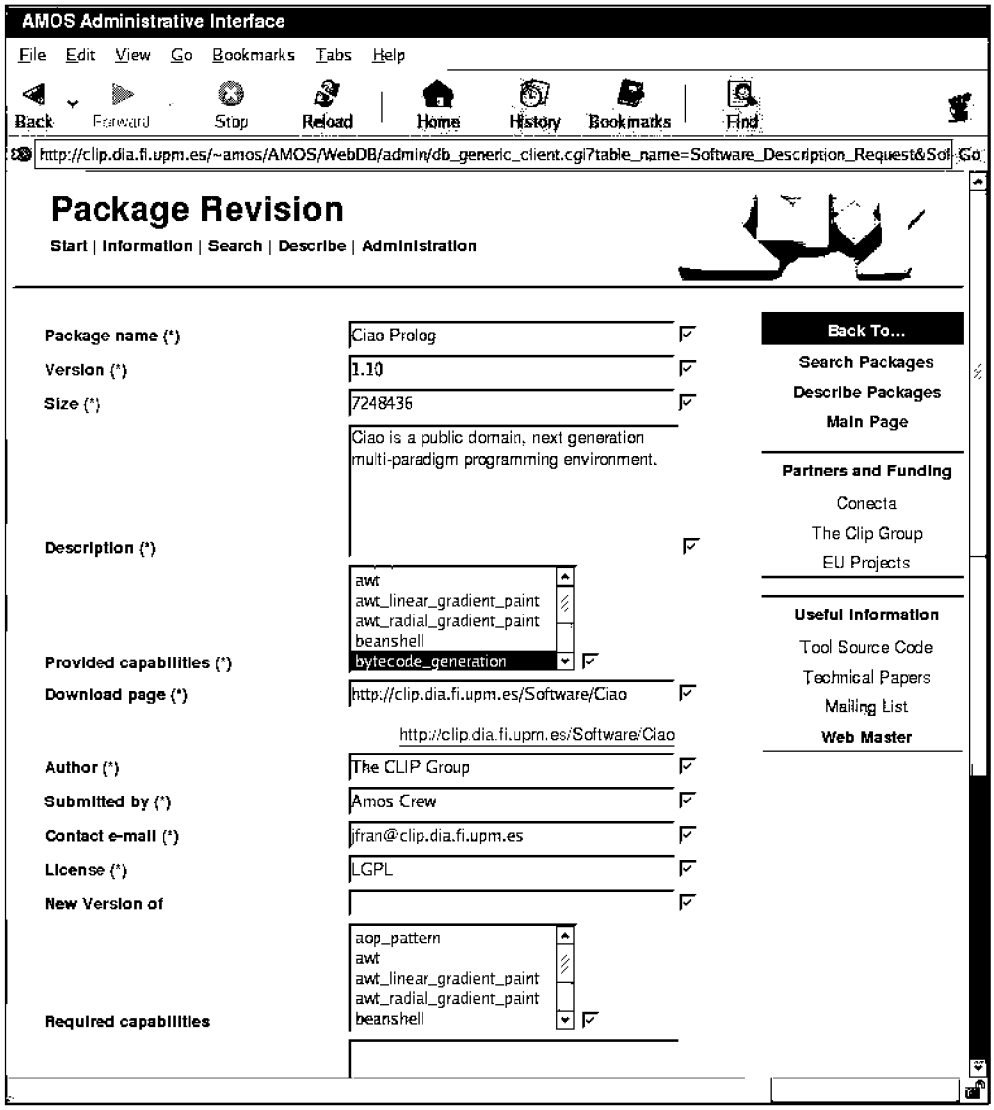

Figure 11: Package validation

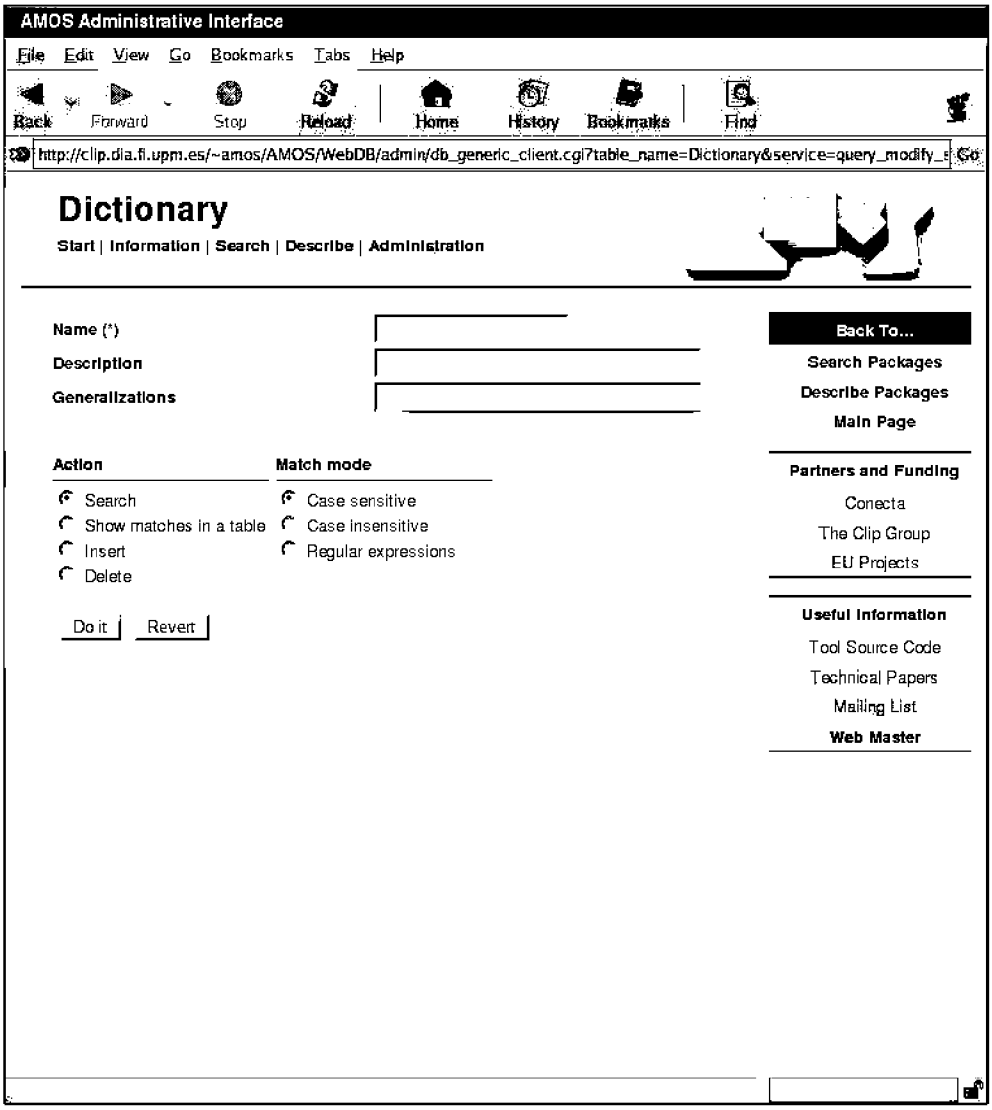

Figure 12: Specialized edition of dictionary entries

7 Accessing the Source Code of the Tool

Public, read-only access to the source files can be done through CVS. Just issue the command:

```
cvs -d \
  :pserver:amospub@clip.dia.fi.upm.es:/home/clip/CvsRoot \
  co Systems/Amos
```

in any shell,[1] or set the corresponding parameters in your favorite tool. This will give you the latest snapshot of the sources, which include this documentation. Note that this applies only to the files in the Amos project, and not to other projects stored in the CVS server of the Clip Group. It does not give you write access, either. Please get in touch with `amos _at_ clip.dia.fi.upm.es` if you want to contribute!

The CVS repository holding the sources (Figure 13) can be accessed by pointing any web browser to
`http://clip.dia.fi.upm.es/ViewCVS/viewcvs.cgi/Systems/Amos/`

[1] Backslashes are meant to represent line changes, and can be omitted if the command and arguments are written in a single line.

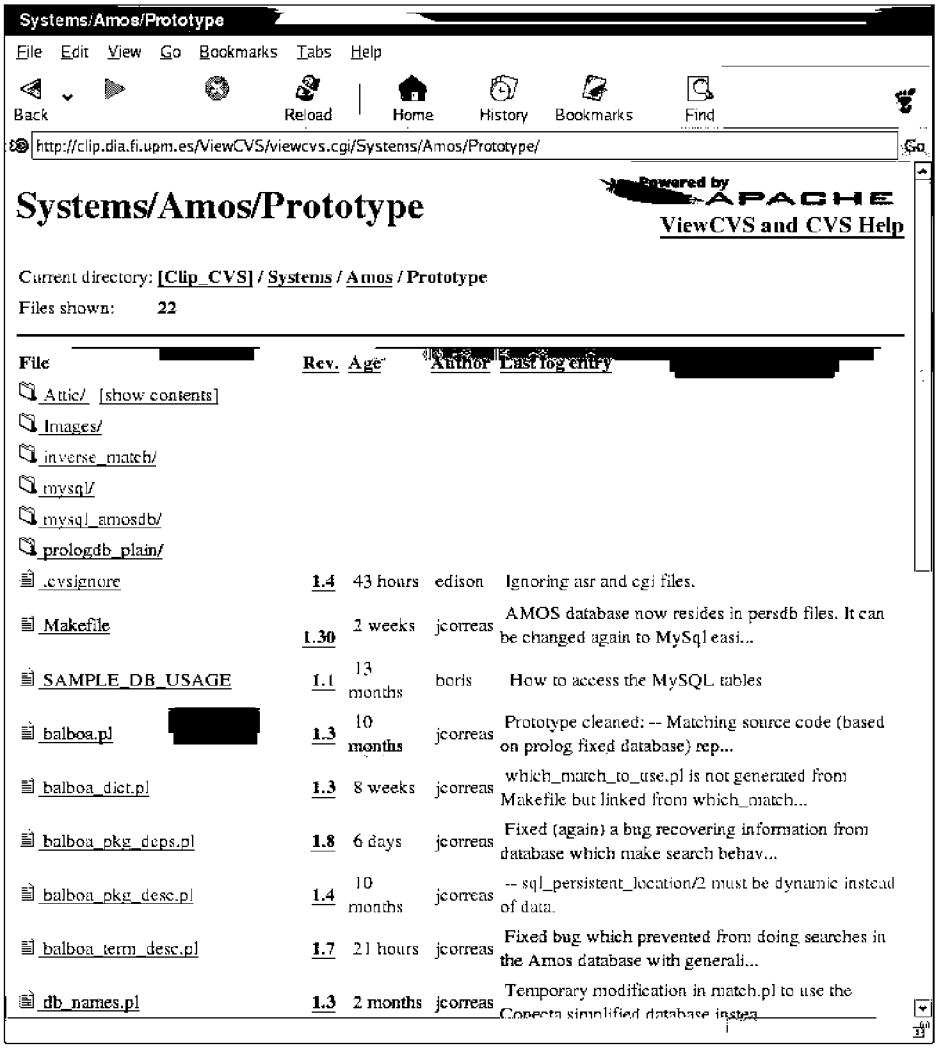

Figure 13: WWW view of the CVS repository

8 Mail List

A public mail list (Figure 14) has been set up at the Conecta mail server, which also hosts a number of other Open Source related lists. It is reachable from

`http://mail.conecta.it/mailman/listinfo/amos-general`

which includes information on how to subscribe.

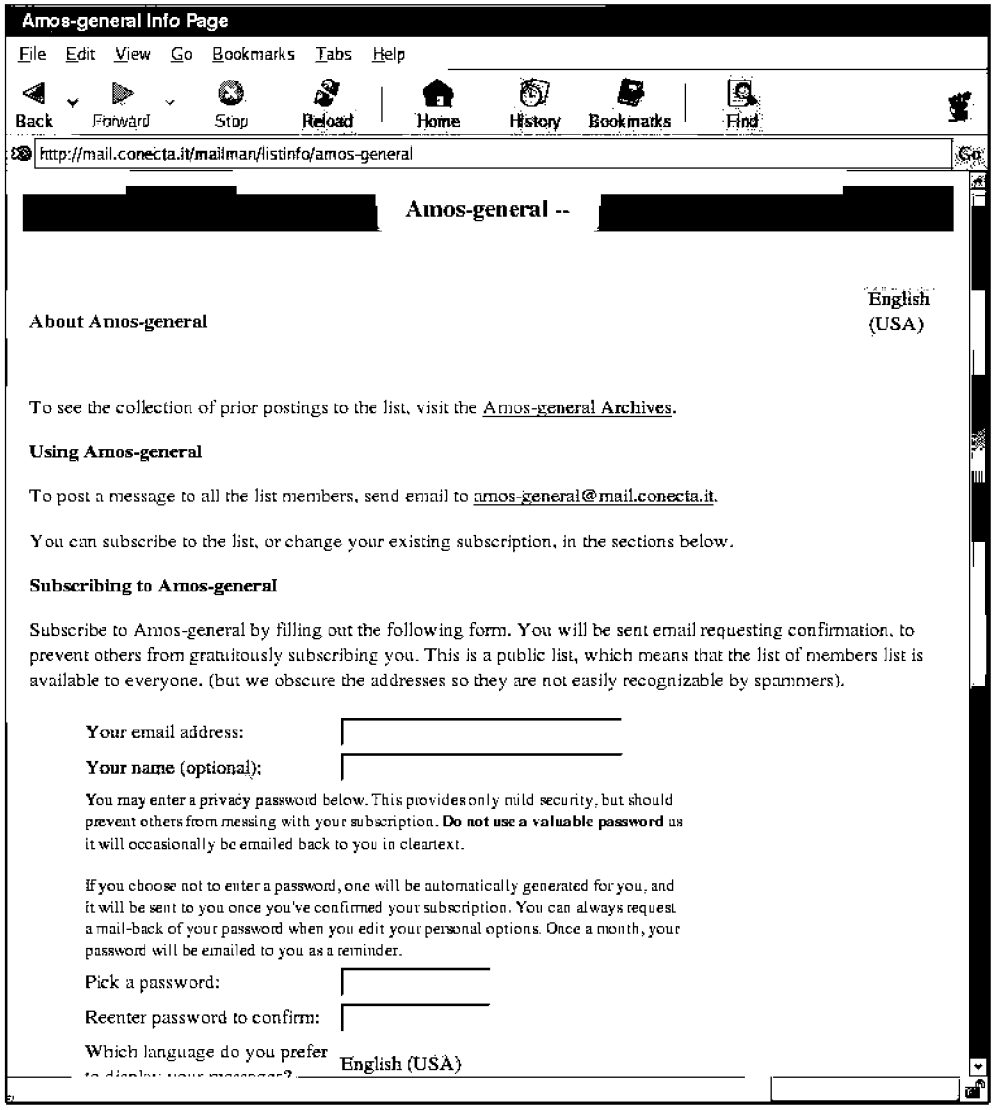

Figure 14: Mail list

10 More Information

More information is available through the "Technical Information" section of the Amos tool. The information currently reachable from there is summarized below. Reading them is recommended for anyone interested in the Amos approach, in extending Amos, or in making a similar tool.

Technical Reports

- Deliverable D02: Ontology[2]
- Deliverable D03: The Internal Query Language[3]

Papers and Talks Related to the Project

- Talk given at the workshop *Free Software and Research*[4], Soissons, France, December 2003.
- Talk given at University Rey Juan Carlos I, Madrid, 2003.[5]
- Project Summary delivered at the Open Source Meeting, Brussels, 2002.[6]
- Paper presented at the V Hispalinux Conference.[7]
- Paper presented at the First CologNet Workshop on Component-Based Software Development and Implementation Technology for Computational Logic Systems.[8]

[2] http://www.amosproject.org/Papers/ontology.pdf
[3] http://www.amosproject.org/Papers/internal_QL_design.pdf
[4] http://clip.dia.fi .upm.es/papers/FSR2003-amos.pdf
[5] http://lml.ls.fi .upm.es/~mcarro/Slides/Free_Software/Amos_Talk_URJC/
[6] http://lml.ls.fi .upm.es/~mcarro/Slides/Free_Software/Amos_Presentation_Brussels/
[7] http://clip.dia.fi .upm.es/papers/amos-hispalinux-TR.pdf
[8] http://clip.dia.fi .upm.es/papers/amos-cbd.pdf

References

[BLHL01] T. Berners-Lee, J. Hender, and O. Lassila. The Semantic Web. *Scientific American*, May 2001. Available from `http://www.sciam.com/2001/0501issue/0501berners-lee.html`.

[CGC+04] M. Carro, J. M. Gomez, J. Correas, J. F. Morales, E. F. Mera, G. Puebla, D. Cabeza, F. Bueno, C. Daffara, and M. Hermenegildo. Web site. Technical Report CLIP9/2004.0, Computer Science School, Technical University of Madrid, School of Computer Science, UPM, May 2004. Deliverable D17 of the AMOS Project.

[Daf02] Carlo Daffara. An ontology for open source code. Technical report, Conecta s.r.l., 2002. Deliverable D2 of the AMOS Project.

[dRG95] Maria del Rosario Girardi. *Classification and Retrieval of Software through their Description in Natural Language*. PhD thesis, Computer Science Department, University of Geneva, 1995.

[GCM04] J. M. Gomez, M. Carro, and J. F. Morales. The external interface. Technical Report CLIP6/2004.0, Computer Science School, Technical University of Madrid, School of Computer Science, UPM, May 2004. Deliverable D12 of the AMOS Project.

[GI95] M. R. Girardi and B. Ibrahim. Using English to Retrieve Software. *The Journal of Systems and Software*, 30(3):249–270, September 1995.

[HBC+99] M. Hermenegildo, F. Bueno, D. Cabeza, M. Carro, M. García de la Banda, P. López-García, and G. Puebla. The CIAO Multi-Dialect Compiler and System: An Experimentation Workbench for Future (C)LP Systems. In *Parallelism and Implementation of Logic and Constraint Logic Programming*, pages 65–85. Nova Science, Commack, NY, USA, April 1999.

[HBC+00] M. Hermenegildo, F. Bueno, D. Cabeza, M. Carro, M. García de la Banda, P. López-García, and G. Puebla. The Ciao Logic Programming Environment: A Tutorial. In *International Conference on Computational Logic, CL2000*, July 2000.

[Mau00] P.M. Maurer. Components: What if the gave a revolution and nobody came? *IEEE Computer*, pages 28–34, June 2000.

[MMM95] H. Mili, F. Mili, and A. Mili. Reusing Software: Issues and research directions. *IEEE Transactions on Software Engineering*, 1995.

[SWCP] The Semantic Web Community Portal. Markup Languages and Ontologies. Available from `http://www.semanticweb.org/knowmarkup.html`.